Noname manuscript No.
(will be inserted by the editor)

Robust Swarm Optimisation for Fuzzy Open Shop Scheduling

Juan José Palacios · Inés González-Rodríguez · Camino R. Vela · Jorge Puente

the date of receipt and acceptance should be inserted later

Abstract In this paper we consider a variant of the open shop problem where task durations are allowed to be uncertain and where uncertainty is modelled using fuzzy numbers. Solutions to this problem are fuzzy schedules, which we argue should be seen as predictive schedules, thus establishing links with the concept of robustness and a measure thereof. We propose a particle swarm optimization (PSO) approach to minimise the schedule's expected makespan, using priorities to represent particle position, as well as a decoding algorithm to generate schedules in a subset of possibly active ones. Our proposal is evaluated on a varied set of several benchmark problems. The experimental study includes a parametric analysis, results of the PSO compared with the state-of-the-art, and an empirical study of the robustness of taking into account uncertainty along the scheduling process.

Keywords open shop scheduling; fuzzy durations; particle swarm optimisation; robustness

1 Introduction

The open shop scheduling problem is a problem with an increasing presence in the scheduling literature and with clear applications in industry—consider for instance testing facilities, where units go through a series of diagnostic tests that need not be performed in a specified order and where different testing equipment is usually required for each test [27]. For a number of machines $m \geq 3$, this problem is NP-complete; in consequence, it is usually tackled via metaheuristic techniques. For makespan minimisation, in [15] two heuristic methods to obtain a list of operation priorities are described and later used in a list-scheduling algorithm; [23] proposes a tabu search algorithm; ant colony optimisation is hybridised with beam search in [3]; [30] proposes a solution based on particle swarm optimisation; ant bee colony optimisation is used in [18] and a hybrid genetic algorithm is proposed in [1]. These last three algorithms conform the state-of-the art for open shop with makespan minimisation.

To enhance the range of applications of scheduling, part of the research is devoted to incorporating the uncertainty and vagueness pervading real-world situations. Part of this uncertainty translates into variability of input data which can be somehow modelled and anticipated, leading to proactive or robust scheduling [16]. The approaches are diverse and, among these, fuzzy sets have been used in a wide variety of ways [6],[5]. Far from being trivial, incorporating uncertainty and extending heuristic strategies to the resulting setting usually requires a significant reformulation of both the problem and solving methods. Some heuristic methods have so far been proposed for fuzzy flow and job shop problems, where uncertain durations are modelled via fuzzy sets; among others, in the last years we find genetic algorithms in [12] and [22], a fuzzy-neural approach in [32], a memetic algorithm combining evolution and local search in [28], swarm-based neighbourhood search in [33] or differential evolution in [17]. However, to the

Juan José Palacios, Camino R. Vela, Jorge Puente
Dep. of Computer Science,
University of Oviedo, (Spain)
E-mail: palaciosjuan@uniovi.es, crvela@uniovi.es, puente@uniovi.es,
http://di002.edv.uniovi.es/iscop

Inés González-Rodríguez
Dep. of Mathematics, Statistics and Computing,
University of Cantabria, (Spain)
E-mail: gonzalezri@unican.es

best of our knowledge, the open shop problem has received little attention in the fuzzy framework: in [21] fuzzy sets are used to represent flexible job start and due dates, a possibilistic mixed-integer linear programming method is proposed in [25] for a multiobjective open shop with setup times, fuzzy processing times and fuzzy due dates and in [26] a genetic algorithm is proposed to solve the open shop with fuzzy durations, and in [11] this genetic algorithm is combined with a local search method.

In the following, we consider the fuzzy open shop problem with expected makespan minimisation, denoted $O|fuzzp_i|E[C_{max}]$ and propose a particle swarm technique to solve it. The rest of the paper is organized as follows. In Section 2 we formulate the problem and associated concepts and introduce a measure of robustness. Then, in Section 3, we describe the main components of the particle swarm optimisation (PSO) algorithm proposed to solve the problem. Section 4 includes a parametric analysis of the PSO, experimental results to evaluate the competitiveness of our proposal and an additional analysis of the usefulness of scheduling with fuzzy durations in order to improve solution robustness. Finally, in Section 5 we summarise the main conclusions and propose ideas for future work.

2 Open Shop Scheduling with Uncertain Durations

The *open shop scheduling problem*, or *OSP* in short, consists in scheduling a set of n jobs $J_1, \ldots, J_n$ to be processed on a set of m physical resources or machines $M_1, \ldots, M_m$. Each job consists of m tasks or operations, each requiring the exclusive use of a different machine for its whole processing time without preemption, i.e. all operations must be processed without interruption. In total, there are $n \times m$ tasks (nm for short), denoted $\{o_{ij}, 1 \leq i \leq n, 1 \leq j \leq m\}$. A solution to this problem is a *schedule* –an allocation of starting times for all tasks– which is *feasible*, in the sense that all constraints hold, and is also optimal according to some criterion. Here, the objective will be minimising the makespan C_{max}, that is, the time lag from the start of the first task until the end of the last one, a problem often denoted $O||C_{max}$ in the literature [14].

2.1 Uncertain Durations

In real-life applications, it is often the case that it is not known in advance the exact time it will take to process one operation and only some uncertain knowledge is available, for instance, an interval of possible durations, or a most likely duration with a certain error margin. Such knowledge can be modelled using a *triangular fuzzy number* or TFN, given by an interval $[n^1, n^3]$ of possible values and a modal value n^2 in it [7]. For a TFN N, denoted $N = (n^1, n^2, n^3)$, the membership function takes the following triangular shape:

$$\mu_N(x) = \begin{cases} \frac{x-n^1}{n^2-n^1} & : n^1 \leq x \leq n^2 \\ \frac{x-n^3}{n^2-n^3} & : n^2 < x \leq n^3 \\ 0 & : x < n^1 \text{ or } n^3 < x \end{cases} \quad (1)$$

In the open shop, we essentially need two operations on processing times (fuzzy numbers), the sum and the maximum. These are obtained by extending the corresponding operations on real numbers using the *Extension Principle*. However, computing the resulting expression is cumbersome, if not intractable. For the sake of simplicity and tractability of numerical calculations, we follow [8] and approximate the results of these operations, evaluating the operation only on the three defining points of each TFN. It turns out that for any pair of TFNs M and N, the approximated sum $M + N \approx (m^1 + n^1, m^2 + n^2, m^3 + n^3)$ coincides with the actual sum of TFNs; this is not necessarily so for the maximum $\max\{M, N\} \approx (\max\{m^1, n^1\}, \max\{m^2, n^2\}, \max\{m^3, n^3\})$, although they have identical support and modal value.

The membership function of a fuzzy number can be interpreted as a possibility distribution on the real numbers. This allows to define its expected value [24], given for a TFN N by $E[N] = \frac{1}{4}(n^1 + 2n^2 + n^3)$. It coincides with the neutral scalar substitute of a fuzzy interval and the centre of gravity of its mean value [6]. It induces a total ordering $\leq_E$ in the set of fuzzy intervals [8], where for any two fuzzy intervals M, N $M \leq_E N$ if and only if $E[M] \leq E[N]$.

2.2 Fuzzy Open Shop Scheduling

If processing times of operations are uncertain and such uncertainty is modelled using TFNs, the resulting schedule is fuzzy in the sense that starting and completion times for each operation and hence the makespan are TFNs, where each TFN can be seen as a possibility distribution on the actual values that the corresponding time may take. However, there is no uncertainty regarding the order in which operations must be processed.

Indeed, a schedule for an open shop problem of size $n \times m$ (n jobs and m machines) may be determined by a priority vector $\pi = (\pi_1, \ldots, \pi_{nm})$ representing a task processing order, where $\forall k, l = 1, \ldots, nm$ $1 \leq \pi_l \leq nm$ and, if $k \neq l$, then $\pi_k \neq \pi_l$, that is, π is a permutation of the set of tasks where each task o_{ij} may be represented

by the number $(i-1)m+j$. The task processing order represented by the priority vector uniquely determines a feasible schedule; it should be understood as expressing partial orderings for every set of tasks requiring the same machine and for every set of tasks requiring the same job.

Let us assume that the processing time p_{ij} of each task o_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$ is a TFN, so the problem may be represented by a matrix of fuzzy processing times $\boldsymbol{p}$ of size $n \times m$. For a given priority vector π and a task o_{ij}, its starting time $S_{ij}(\pi, \boldsymbol{p})$ is the maximum between the completion times of the task preceding o_{ij} in its job according to π, let it be denoted o_{ik}, and the task preceding o_{ij} in its machine according to π, let it be denoted o_{lj}:

$$S_{ij}(\pi, \boldsymbol{p}) = \max(C_{ik}(\pi, \boldsymbol{p}), C_{lj}(\pi, \boldsymbol{p})) \tag{2}$$

where $C_{ik}(\pi, \boldsymbol{p})$ or $C_{lj}(\pi, \boldsymbol{p})$ are taken to be zero if o_{ij} is the first task to be processed either in its job or its machine. Then, its completion time $C_{ij}(\pi, \boldsymbol{p})$ is obtained by adding its duration p_{ij} to $S_{ij}(\pi, \boldsymbol{p})$:

$$C_{ij}(\pi, \boldsymbol{p}) = S_{ij}(\pi, \boldsymbol{p}) + p_{ij} \tag{3}$$

The completion time of a job J_i will then be the maximum completion time of all its tasks, that is,

$$C_i(\pi, \boldsymbol{p}) = \max_{1 \le j \le m} \{C_{ij}(\pi, \boldsymbol{p})\} \tag{4}$$

so the *fuzzy makespan* $C_{max}(\pi, \boldsymbol{p})$ will be given by the following:

$$C_{max}(\pi, \boldsymbol{p}) = \max_{1 \le i \le n} (C_i(\pi, \boldsymbol{p})) \tag{5}$$

In the case where no confusion is possible, we may drop the priority vector π and the processing times matrix $\boldsymbol{p}$ and simply write C_{max}.

An important issue with fuzzy times is to decide on the meaning of "optimal makespan". It is not trivial to optimise a fuzzy makespan, since neither the maximum nor its approximation define a total ordering in the set of TFNs. Using ideas similar to stochastic scheduling, we follow the approach taken for the fuzzy job shop in [13]. Given the total ordering provided by the expected value, we consider that the objective is to minimise the expected makespan $E[C_{max}]$. The resulting problem may be denoted $O|fuzzp_i|E[C_{max}]$ using the three-field notation [14].

Let us illustrate the previous definitions with an example. Consider a problem of 3 jobs and 2 machines with the following matrix for fuzzy processing times:

$$\boldsymbol{p} = \begin{pmatrix} (3,4,7) & (3,4,7) \\ (2,3,3) & (4,5,6) \\ (3,4,6) & (1,2,4) \end{pmatrix}$$

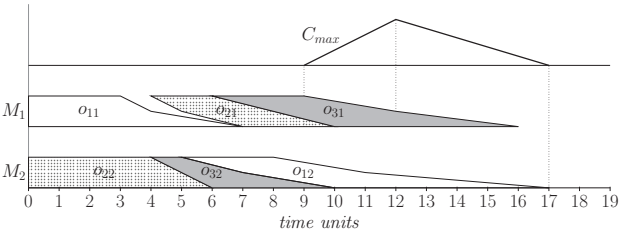

Fig. 1 Gantt chart of the schedule represented by the priority vector (1, 4, 6, 3, 5, 2).

Here $p_{21} = (2,3,3)$ is the processing time of task o_{21}, the task of job J_2 to be processed in machine M_1. Figure 1 shows the Gantt chart adapted to TFNs of the schedule given by the priority vector π=(1, 4, 6, 3, 5, 2). It represents the partial schedules on each machine obtained from this decision variable. Tasks must be processed in the following order: $o_{11}, o_{22}, o_{32}, o_{21}, o_{31}, o_{12}$. Given this ordering, the starting time for task o_{21} will be the maximum of the completion times of o_{22} and o_{11}, which are respectively the preceding tasks in the job and in the machine:

$$S_{21} = \max(C_{22}, C_{11}) = \max((4,5,6),(3,4,7)) = (4,5,7).$$

Consequently, its completion time will be

$$C_{21} = S_{21} + p_{21} = (4,5,7) + (2,3,3) = (6,8,10).$$

Also, it is easy to see that the makespan is $C_{max} = (9,12,17)$, so $E[C_{max}] = 12.5$.

2.3 Robust schedules

A fuzzy schedule does not provide exact starting times for each task. Instead, it gives a fuzzy interval of possible values for each starting time, provided that tasks are executed in the order determined by the schedule. In fact, it is impossible to predict what the exact time-schedule will be, because it depends on the realisation of the tasks durations, which is not known yet. This idea is the basis for a semantics for fuzzy schedules from [12] by which solutions to the fuzzy open shop should be understood as a-priori solutions, also called baseline or predictive schedules in the literature [16]. These solutions are found when the duration of tasks is not exactly known and a set of possible scenarios must be taken into account. When tasks are executed according to the ordering provided by the fuzzy schedule we shall know their real duration and, hence, obtain a real (executed) schedule, the a-posteriori solution with deterministic times.

Clearly, fuzzy solution should yield reasonably good executed schedules in the moment of its practical use.

Also, the estimates for starting and completion times and, in particular, for the makespan, should be reasonably accurate for each possible scenario of task durations. This leads us to the concept of solution robustness. As [19] puts it, "Intuitively, a solution can be considered as robust if it behaves "well" or "not too bad" in all the scenarios.". This is the idea underlying a definition of ϵ-robustness given in [2] for stochastic scheduling which can be adapted to the fuzzy open shop as follows.

A predictive schedule is considered to be *robust* if the quality of the eventually executed schedule is close to the quality of the predictive schedule. In particular, a predictive schedule with objective value f^{pred} is ϵ-*robust* for a given ϵ if the objective value f_{exec} of the eventually executed schedule is such that:

$$(1-\epsilon) \leq \frac{f^{exec}}{f^{pred}} \leq (1+\epsilon) \tag{6}$$

or, equivalently,

$$\frac{|f^{exec} - f^{pred}|}{f^{pred}} \leq \epsilon \tag{7}$$

That is, the relative error of the estimation made by the predictive schedule is bounded by ϵ. Obviously, the smaller ϵ is, the better.

3 Particle Swarm Optimization for the FOSP

Given the complexity of the open shop, different metaheuristic techniques have been proposed to solve the general m-machine problem. In particular, a method based on particle swarm optimisation has been proposed in [30] which is considered the state-of-the-art for crisp open shop.

Require: A FOSP instance
Ensure: A schedule for the input instance
1. generate and evaluate the initial swarm.
2. compute gbest and pbest for each particle.
while no termination criterion is satisfied **do**
  for each particle k **do**
    for each dimension d **do**
      3. update velocity v_d^k.
      4. update position x_d^k.
    5. evaluate particle k.
    6. update pbest and gbest values.
return The schedule from the best particle evaluated so far;

Alg. 1: A generic PSO algorithm

Particle Swarm Optimisation (PSO) is a population-based stochastic optimisation technique inspired by bird flocking or fish schooling [20]. In PSO, each position in the search space corresponds to a solution of the problem and particles in the swarm cooperate to find the best position (hence best solution) in the space. Particle movement is mainly affected by the three following factors:

- Inertia: Velocity of the particle in the latest iteration.
- *pbest*: The best position found by the particle.
- *gbest*: The best position found by the swarm so far (the best *pbest*).

The potential solutions or particles fly through the problem space changing their position and velocity by following the current optimum particles *pbest* and *gbest*.

Algorithm 1 describes the structure of a generic PSO algorithm. First, the initial swarm is generated and evaluated. Then the swarm evolves until a termination criterion is satisfied and in each iteration, a new swarm is built from the previous one by changing the position and velocity of each particle following its *pbest* and *gbest* locations.

Following this general structure, we now extend the successful algorithm from [30] to the fuzzy framework.

3.1 Position Representation and Evaluation

We use a priority-based representation for particle positions. Thus a schedule is encoded as a priority matrix $X^k = (x_{ij}^k)_{i=1\ldots n, j=1\ldots m}$, where x_{ij}^k denotes the priority of operation o_{ij}, the task of job i processed on machine j. An operation with smaller x_{ij}^k has a higher priority to be scheduled.

If we represent a FOSP solution as a task processing order π, which is a permutation of tasks, we can transfer this permutation to a priority matrix and viceversa. For instance, given the following solution for a problem of size 3×3:

$$\pi = \left(o_{11}\ o_{13}\ o_{23}\ o_{12}\ o_{31}\ o_{33}\ o_{21}\ o_{32}\ o_{22} \right)$$

a particle in the space can be obtained by randomly setting x_{ij} in the interval $(p-0.5, p+0.5)$ where p is the location of o_{ij} in π. Therefore, the operation with smaller x_{ij} has higher priority to be scheduled. The above permutation list can be transferred to:

$$X^k = \begin{pmatrix} 1.2 & 4.0 & 1.7 \\ 6.6 & 9.4 & 2.7 \\ 5.3 & 7.9 & 6.4 \end{pmatrix}$$

Decoding of a particle may be done in different ways. For the crisp job shop and by extension for the open shop, it is common to use the G&T algorithm [9], which is an active schedule builder. A schedule is *active* if one

task must be delayed for any other one to start earlier. Active schedules are good in average and, most importantly, the space of active schedules contains at least an optimal one, that is, the set of active schedules is *dominant*. For these reasons it is worth to restrict the search to this space. In [10] a narrowing mechanism was incorporated to the G&T algorithm in order to limit machine idle times by means of a delay parameter $\delta \in [0, 1]$, thus searching over the space of so-called parameterised active schedules. In the deterministic case, for $\delta < 1$ the search space is reduced so it may no longer contain optimal schedules and, at the extreme $\delta = 0$ the search is constrained to *non-delay* schedules, where a resource is never idle if a requiring operation is available. This variant of G&T has been applied in [30] to the deterministic OSP, under the name "parameterized active schedule generation algorithm".

In Algorithm 2 we propose an extension of parameterised G&T to the case of fuzzy processing times, denoted $pfG\&T$. It should be noted that, due to the uncertainty in task durations, even for $\delta = 1$, we cannot guarantee that the produced schedule will indeed be active when it is actually performed (and tasks have exact durations). We may only say that the obtained fuzzy schedule is *possibly active*. Throughout the algorithm, Ω denotes the set of the operations that have not been yet scheduled, X^k the priority matrix, S_{ij} the starting time of the operation o_{ij} and C_{ij} the completion time of the operation o_{ij}.

Let us illustrate the decoding algorithm with an example. Consider the problem proposed in subsection 2.2 to illustrate the concept of fuzzy schedule and the following priority matrix for it:

$$X^k = \begin{pmatrix} 1.2 & 5.3 \\ 2.7 & 1.7 \\ 4.0 & 6.4 \end{pmatrix}$$

Figure 2 shows the Gantt chart of the partial schedule in which the operations o_{11}, o_{22} and o_{21} have been already scheduled. In this situation, $\Omega = \{o_{12}, o_{31}, o_{32}\}$. Table 1 depicts the values of the starting and completion times of the operations in Ω in this iteration of the algorithm as well as its expected values. The s^* and c^* values are shown in bold. Considering $\delta = 1$, the conflict set is $O = \{o_{12}, o_{32}\}$. Operation o_{31} is not contained in the O set, although it has the highest priority in Ω, because the possibility that operation o_{32} can be completed before the earliest beginning of o_{31} is 1, so by selecting o_{31} before o_{31} we would generate a non possibly active schedule. Additionally, reducing the δ value to 0.1, the set of operations that are candidates to be scheduled is further restricted to $O = \{o_{32}\}$ even though o_{32} is the lowest-priority operation of Ω.

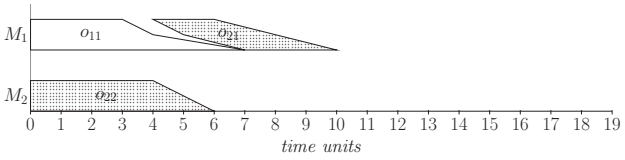

Fig. 2 Gantt chart of the partial schedule for the example in Section 2.2 where only o_{11}, o_{22} and o_{21} have been scheduled.

Table 1 Partial schedule values.

Oper.	S_{ij}	$E[S_{ij}]$	C_{ij}	$E[C_{ij}]$
o_{12}	(4,5,7)	5.25	(7,9,14)	9.75
o_{31}	(6,8,10)	8.00	(9,12,16)	12.25
o_{32}	(4,5,6)	**5.00**	(5,7,10)	**7.25**

Notice that the $pfG\&T$ algorithm only uses the priority vector to break ties among tasks in the conflict set, so the task processing order in the resulting schedule may differ from that in the particle. Given that the essence of a particle is the task ordering it represents, *gbest* and *pbest* do not record the actual best positions found so far, but rather the best operation sequences of the schedules generated by the decoding operator.

Require: A FOSP instance and a particle position X^k
Ensure: A schedule for the input instance considering the priorities given by X^k
  $\Omega \leftarrow \{o_{ij} : 1 \leq i \leq n, 1 \leq j \leq m\}$;
  while $\Omega \neq \emptyset$ **do**
    $c^* \leftarrow \min_{o_{ij} \in \Omega}\{E[C_{ij}]\}$;
    $s^* \leftarrow \min_{o_{ij} \in \Omega}\{E[S_{ij}]\}$;
    $O \leftarrow \{o_{ij} : E[S_{ij}] < s^* + \delta \times (c^* - s^*), o_{ij} \in \Omega\}$;
    Choose the operation o^*_{ij} from O with smallest x^k_{ij};
    Schedule the operation o^*_{ij};
    $\Omega \leftarrow \Omega - \{o^*_{ij}\}$;
  return The schedule given by $\{S_{ij} : 1 \leq i \leq n, 1 \leq j \leq m\}$

Alg. 2: The $pfG\&T$ algorithm

3.2 Particle movement and velocity

Particle movement depends not only on its position, but also on its velocity. For any particle, its velocity is represented by an array of the same length as the position array where all the values are in the set $\{-1, 0, 1\}$. Initially, the values in the array are set at random. Afterwards, particle position and velocity are updated depending on *gbest* and *pbest*. Traditionally, this updating depends on distance values. Instead, this PSO considers whether the position value x^k_{ij} is larger or smaller than $pbest^k_{ij}$ ($gbest_{ij}$). Updating is controlled at the beginning of each iteration by the inertia weight w as well as two other constants $0 \leq C_1, C_2$ such that $C_1 + C_2 \leq 1$, representing the probability that the updating is guided

either by *pbest* or by *gbest*. Further detail on the updating process can be found in Algorithm 3.

Require: A particle position X^k and velocity V^k, best particle and swarm positions $pbest^k$ and $gbest$, inertia w and updating probabilities C_1, C_2
Ensure: The updated particle position X^k and velocity V^k
for each dimension d **do**
generate random value $rand \sim U(0,1)$.
if $v_d^k \neq 0$ and $rand \geq w$ **then**
$v_d^k \leftarrow 0$.
if $v_d^k = 0$ **then**
generate random value $rand \sim U(0,1)$.
if $rand \leq C_1$ **then**
if $pbest_d^k \geq x_d^k$ **then**
$v_d^k \leftarrow 1$.
else
$v_d^k \leftarrow -1$.
generate random value $rand_2 \sim U(0,1)$.
$x_d^k \leftarrow pbest_d^k + rand_2 - 0.5$.
if $C_1 < rand \leq C_1 + C_2$ **then**
if $gbest_d \geq x_d^k$ **then**
$v_d^k \leftarrow 1$.
else
$v_d^k \leftarrow -1$.
generate random value $rand_2 \sim U(0,1)$.
$x_d^k \leftarrow gbest_d + rand_2 - 0.5$.
else
$x_d^k \leftarrow x_d^k + v_d^k$.
return The updated particle position X^k and velocity V^k;

Alg. 3: Particle movement

Position mutation. In order to introduce diversity, after a particle moves to a new position, we mutate it with probability p_M by choosing an operation and then randomly changing its priority value x_d^k independently of v_d^k . As in [30], for a problem of size $n \times m$, if $x_d^k < (nm/2)$, x_d^k will take a random value in $[mn - n, mn]$, and $v_d^k = 1$; else, if $x_d^k > (nm/2)$, x_d^k will take a random value in $[0, n]$ and $v_d^k = -1$.

Diversification strategy. If all particles have the same *pbest* solutions, they will be trapped into local optima. To prevent such situation, a diversification strategy is adopted that keeps the *pbest* solutions different. In this strategy, the *pbest* solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far, where N is the size of the swarm. Once any particle generates a new solution, the *pbest* solutions will be updated in certain cases as follows:

- if the makespan of the particle solution equals that of any *pbest* solution, replace that *pbest* solution with the new particle solution;
- if the makespan of the particle solution improves the worst *pbest* solution and is different from all *pbest* solutions, set the worst *pbest* solution to be the particle solution.

4 Experimental evaluation

We now proceed to empirically evaluate the proposed method in several steps, using a total of 520 problem instances. First, a parametric analysis will be conducted to decide on a good parameter-configuration for the PSO search process as well as for the schedule generation algorithm $pG\&T$. Then, we present results of expected makespan minimisation obtained by the PSO and we compare them with the best results obtained so far in the literature by a memetic algorithm. Finally, we shall present some results to illustrate the benefits in terms of robustness of using fuzzy numbers along the scheduling process, instead of the more straightforward approach of scheduling a crisp problem that results from defuzzification.

4.1 Experiment setting

For the experimental study we use the test bed given in [11], where the authors follow [8] and generate a set of fuzzy problem instances from well-known open shop benchmark problems [4]. Given a crisp problem instance, each crisp processing time t is transformed into a symmetric fuzzy processing time $p(t)$ such that its modal value is $p^2 = t$ and p^1, p^3 are random values, symmetric w.r.t. p^2 and generated so the TFN's maximum range of fuzziness is 30% of p^2. The original problem instances consist of 6 families, denoted *J3, J4,..., J8*, of sizes 3×3 to 8×8, containing 8 or 9 instances each. Ten fuzzy versions of each crisp problem instance were generated, so in total there are 520 problem instances. The obtained benchmark instances for the fuzzy open shop are available at `http://www.di.uniovi.es/iscop`.

If a lower bound for the expected makespan were known, the algorithm's performance may be evaluated by measuring the distance between the obtained expected makespan and the lower bound. Thanks to the symmetry in the TFNs, the optimal solution (if known) to the crisp problem provides a lower bound, denoted LB_c, for the expected makespan of the fuzzified version [8]. An alternative lower bound LB_f can be obtained directly from the fuzzy instance as follows:

$$LB_f = E[\max\{\max_j\{\sum_{i=1}^{n} p_{ij}\}, \max_i\{\sum_{j=1}^{m} p_{ij}\}\}] \quad (8)$$

This lower bound adapts the lower bound proposed in [31] for crisp problems to the fuzzy setting. The maximum of both quantities yield a tighter lower bound for the expected makespan of the optimal solution:

$$LB = \begin{cases} \max(LB_f, LB_c) \text{ if } LB_c \text{ is known} \\ LB_f \text{ otherwise} \end{cases} \quad (9)$$

We can now compute the makespan relative error with respect to LB as follows:

$$RE = \frac{E[C_{max}] - LB}{LB} \quad (10)$$

This relative error will be the basis for evaluating the obtained results in the remaining of this section.

All the experiments reported in this section, correspond to a C++ implementation running on a PC with Xeon E5520 processor and 24 GB RAM running Linux (SL 6.0).

4.2 Parametric analysis

For the PSO we take as initial parameter configuration the values proposed after a parameter analysis for the crisp OSP in [30]: swarm size $N = 60$, $C_1 = 0.7$, $C_2 = 0.1$, mutation probability $P_M = 1$ and inertia weight w linearly decreasing from 0.9 to 0.3. Regarding the filtering mechanism of the search space given in the schedule generator $pfG\&T$, an initial value of $\delta = 0.25$ is adopted. This has been done after some preliminary experiments consisting in generating random solutions in the search space with varying values of δ and adopting the value with better solutions in average.

Starting with this initial configuration, we proceed to perform a parametric analysis for both the PSO algorithm and the decoding scheme. First, we try different values for the PSO algorithm's parameters (in bold we highlight the values of the starting configuration) as follows:

- Stopping criterion: A maximum number of iterations $MaxIter$ less or equal than **5000** iterations.
- Guiding constants C_1 and C_2: all possible pairs of the values in $\{\mathbf{0.1}, 0.3, 0.5, \mathbf{0.7}, 0.9\}$, provided that they add up to a maximum of 1.
- Inertia: Linearly decreasing from ω_s to ω_e, with $\omega_s \in \{0.5, 0.7, \mathbf{0.9}\}$ and $\omega_e \in \{0.1, \mathbf{0.3}, 0.5\}$.
- Mutation probability: Values in $\{0, 0.25, 0.50, 0.75, \mathbf{1}\}$.
- Swarm Size N: Values in $\{\mathbf{60}, 80, 100\}$.

We follow an incremental process in which we test one of the parameters until it is optimised, then we fix it and proceed in the same way with the next one, until all the parameters are fixed.

Table 2 Final value for the stopping criterion $MaxIter$ depending on problem size.

Size	$MaxIter$
3×3	100
4×4	100
5×5	750
6×6	1500
7×7	2100
8×8	2700

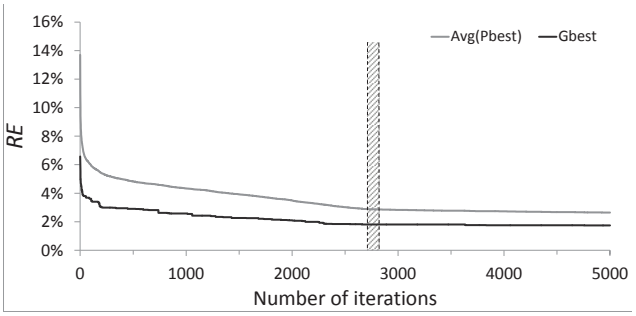

Fig. 3 Evolution along 5000 iterations of $E[C_{max}]$ for *gbest* (in black) and the average $E[C_{max}]$ for *pbest* (in grey) for *j8-per10-1* instance; convergence is obtained at iteration 2700.

4.2.1 Stopping Criterion MaxIter

First, for each problem size we estimate the number of iterations $MaxIter$ needed by the algorithm to converge. The procedure to obtain this number of iterations is as follows: For an arbitrary fuzzy instance of each original crisp problem, the algorithm is run 10 times for 5000 iterations. At each iteration, we record the average makespan of all *pbest* elements in the swarm and calculate its relative error w.r.t. the lower bound LB. Then, we pick the first iteration for which this error will decrease less than 1% in the next 100 iterations. This provides us with an stopping iteration for each problem. Putting together all problems of the same size, we select the number of iterations in the third quartile as $MaxIter$ for the group of problem instances of the same size.The resulting values for $MaxIter$ depending on problem size can be seen in Table 2. Additionally, Figure 3 shows the evolution of the expected makespan for *gbest* and the average expected makespan for *pbest* particles along 5000 iterations of the PSO for problem instance $J8-per10-1$; it highlights the first interval of 100 iterations where the error improvement is less than 1%.

4.2.2 Guiding constants C_1 and C_2

To decide on the values of the remaining parameters, at each step, the PSO is run 10 times on a random fuzzy instance from each 8×8 original problem and

the quality of the configuration used is measured using RE, the relative error w.r.t. the problem's lower bound LB, averaged across the 10 runs. First, we test the guiding probabilities C_1 and C_2. Table 3 shows the average across the 8×8 problems of the relative error RE. Two configurations (in bold in the table) perform clearly better than the rest: $(C_1, C_2) = (0.7, 0.1)$ and $(C_1, C_2) = (0.9, 0.1)$. Since the latter yields slightly better results, we take it to be the definite one.

Table 3 Algorithm's performance with varying guiding constants C_1 and C_2.

Values	RE
$C_1 = 0.1, C_2 = 0.1$	2.96%
$C_1 = 0.1, C_2 = 0.3$	2.83%
$C_1 = 0.1, C_2 = 0.5$	2.81%
$C_1 = 0.1, C_2 = 0.7$	2.79%
$C_1 = 0.1, C_2 = 0.9$	2.92%
$C_1 = 0.3, C_2 = 0.1$	2.63%
$C_1 = 0.3, C_2 = 0.3$	2.74%
$C_1 = 0.3, C_2 = 0.5$	2.77%
$C_1 = 0.3, C_2 = 0.7$	2.80%
$C_1 = 0.5, C_2 = 0.1$	2.64%
$C_1 = 0.5, C_2 = 0.3$	2.64%
$C_1 = 0.5, C_2 = 0.5$	2.74%
$\boldsymbol{C_1 = 0.7, C_2 = 0.1}$	**2.58**%
$C_1 = 0.7, C_2 = 0.3$	2.65%
$\boldsymbol{C_1 = 0.9, C_2 = 0.1}$	**2.56%**

4.3 Inertia Weight Bounds w_s and w_e

Regarding the inertia parameter, Figure 4 shows the average RE obtained using different inertia values linearly decreasing in the corresponding intervals $[w_s, w_e]$ on the X-axis. As above there are two configurations ($w_s = 0.9$ to $w_e = 0.3$ and $w_s = 0.5$ to $w_e = 0.1$) behaving similarly, with slightly better results for the inertia going from 0.9 to 0.3. However, since there are two configurations which are similar in terms of quality both for the guiding constants and the inertia, we have additionally tested all the combinations of those values. Table 4 shows the obtained results, which support taking $C_1 = 0.9, C_2 = 0.1$ and w linearly decreasing in $[0.9, 0.3]$.

Table 4 Algorithm's performance depending on different combinations of guiding constants and inertia weights.

Guiding Constants	Inertia Weights w	
	$0.5 \to 0.1$	$0.9 \to 0.3$
$C_1 = 0.7$, $C_2 = 0.1$	2.590%	2.580%
$C_1 = 0.9$, $C_2 = 0.1$	2.559%	**2.557**%

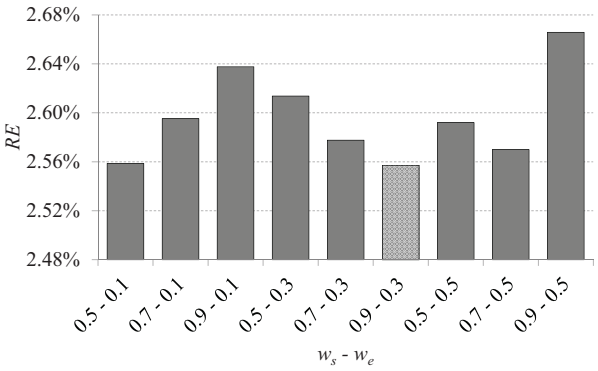

Fig. 4 Algorithm's performance with varying inertia weight from w_s to w_e.

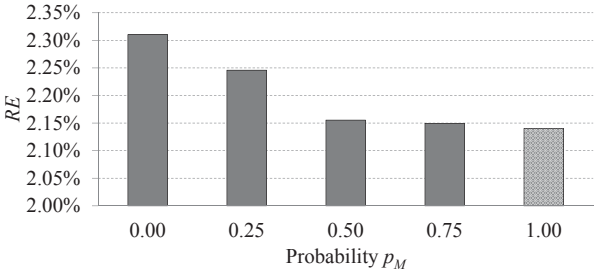

Fig. 5 Algorithm's performance with varying mutation probability.

4.3.1 Mutation Probability p_M

Having fixed the inertia, we try different mutation probability values p_M. Figure 5 illustrates the importance of this parameter for the behaviour of the algorithm, with larger probability values yielding the best results. As it happens in the crisp version of the PSO, the best option is to mutate the particles with probability $p_M = 1$.

4.3.2 Swarm Size N

Finally, we test different swarm sizes: 60, 80 and 100. Here we need to pay attention not only to makespan values, but also to runtime, this being an important penalisation factor. Figure 6 shows the distance (bars) and runtime (line) of each configuration. Clearly, although the largest swarm size provides the best results in terms of relative error w.r.t. the lower bound, the improvement does not compensate the increased computation-time cost. In fact, if we increase the swarm size from 60 to 100, RE decreases 4.3% compared to a 67.3% increase in runtime. Therefore, we keep $N = 60$.

4.3.3 Delay Parameter δ

There is another parameter in the algorithm, the delay parameter δ used by the schedule builder (Algorithm 2). Unlike the other parameters, the differences

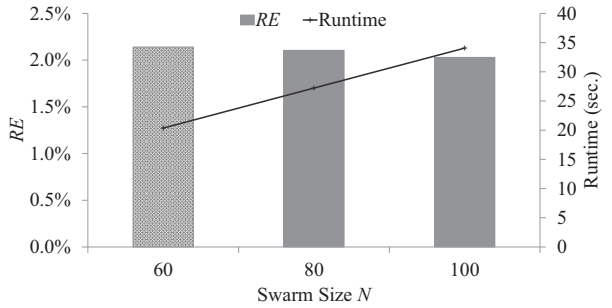

Fig. 6 Algorithm's performance with varying swarm size N.

in algorithm performance caused by variations of δ are a consequence of changes in the subset of the search space which is explored, not of changes in the search process followed by the PSO algorithm. We have tried five possibilities: $\delta = 0$, which corresponds to exploring the set of non-delay schedules; $\delta = 1$, to exploring the set of active schedules; and $\delta = 0.25, 0.5, 0.75$, to exploring three proper subsets of the space of active schedules. Table 5 reports the obtained results, suggesting that the best performance of the PSO is obtained with $\delta = 0.25$.

Table 5 Algorithm's performance with varying the delay parameter δ.

Delay value δ	RE	Std.Dev
0.00	3.44%	2.32
0.25	2.56%	1.96
0.50	2.73%	2.02
0.75	3.32%	2.26
1.00	5.95%	3.38

A summary of the parameter values (except $MaxIter$) adopted after the parametric analysis can be seen in Table 6.

4.4 Algorithm's Evaluation

To our knowledge, the best results obtained so far for the FOSP have been published in [11]. In that paper, the GA from [26] is combined with local search using a new neighbourhood structure, providing a memetic algorithm (MA) which not only improves on solution quality but is also more "reliable" in the sense that there is less variability in solution quality across different executions. Thus, in the following we shall evaluate our PSO in comparison with this MA.

Table 6 Parameter values adopted after the parametric analysis.

Inertia w	Mutation p_m	Guiding const. C_1, C_2	Delay δ
from 0.9 to 0.3	1	0.9, 0.1	0.25

Table 7 Comparison between PSO and MA

Problem Family	PSO RE AoB	PSO RE AoA	PSO $T(s)$	MA RE AoB	MA RE AoA	MA $T(s)$
J3	0.112	0.112	0.05	0.112	0.112	0.13
J4	0.645	0.757	0.10	0.645	0.799	0.23
J5	0.667	0.687	1.29	0.874	2.234	1.29
J6	0.861	1.019	4.24	0.929	2.698	7.57
J7	1.591	1.971	9.76	2.425	4.710	14.46
J8	2.051	2.693	19.53	3.565	5.807	26.15

For the experimental evaluation, we shall use the optimal configuration obtained above with the exception of the smallest problems (3×3, 4×4). Here, since the search space is small, we take $\delta = 1$ as delay value for the $pfG\&T$ algorithm. This allows to explore the whole space of active schedules, thus keeping the chance of finding an optimal solution. For medium size problems (5×5, 6×6) and large problems (7×7, 8×8), we use the best delay value found during the parametric analysis, that is, $\delta = 0.25$.

To evaluate the performance, we run the proposed PSO 30 times for each problem instance, recording the best and average relative error of the expected makespan with respect to its lower bound across these 30 runs. Table 7 contains a summary of the results, with average values across 30 executions on each the 80–90 instances of the same size (detailed results for each problem, which require 520 rows, can be found at `http://www.di.uniovi.es/iscop`). There are three columns per method, PSO and MA, showing the Average of the Best values (AoB), the Average of the Average values (AoA), and the average runtime in seconds ($T(s)$) across 30 runs. We can see that the performance of both PSO and MA is similar on the small (3×3, 4×4) problem instances. However, differences in solution quality between the PSO and the MA increase with problem size, with the PSO obtaining better results. The average increase in RE value from the PSO to the MA across all problems is 28% for AoB and 108% for AoA. It is noticeable that the a maximum increase in the value of AoB in Table 7 is 74% for problems of size 8×8.

More detailed results are presented in Table 8, where each row corresponds to a set of ten fuzzy versions of each of the original crisp problems of size 8×8. It shows relative makespan errors w.r.t. the lower bound for both methods: the best (B) error and the average (A) error across 30 runs of each method. As expected, the PSO compares favourably with MA in all instances. Notice as well that the relative errors for the best (B) and av-

Table 8 Average *RE* (in %) for sets of problems of size 8×8.

	PSO		MA	
Problem	*B*	*A*	*B*	*A*
J8-per0-1	4.410	5.421	7.533	10.493
J8-per0-2	5.402	5.909	6.923	9.715
J8-per10-0	2.951	3.425	4.209	6.688
J8-per10-1	1.614	2.375	3.419	5.959
J8-per10-2	1.352	2.650	3.994	6.455
J8-per20-0	0.393	0.872	1.170	2.960
J8-per20-1	0.000	0.111	0.155	1.349
J8-per20-2	0.288	0.783	1.114	2.837

erage (A) solution do not differ greatly, suggesting that the PSO is quite stable. Figure 7 depicts the average relative makespan error for each set of fuzzy problems, clearly illustrating the difference between the proposed PSO and the MA.

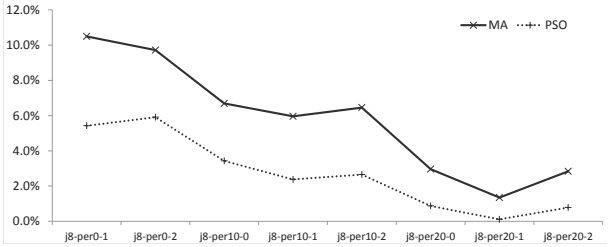

Fig. 7 Average makespan error *RE* (%) for 8×8 problems.

4.5 Why Not Simply Defuzzify?

It is tempting to think that a simpler approach to fuzzy open shop problems is to use defuzzification: substitute the uncertain durations for a crisp value (e.g. their expected value) and then solve the resulting deterministic open shop problem. This would provide a deterministic predictive schedule, including a task processing order. Tasks can then be processed according to this order, even if their starting times are likely to change given the variability in the durations. The advantages of doing so are clear: a simpler operational setting and the availability of different solving methods from the literature. However, before embracing defuzzification, we should also consider its effect (if any) on the robustness of the obtained solutions.

In this subsection, we propose to evaluate, in terms of ϵ-robustness, the predictive schedules obtained with both approaches: solving the fuzzy problem or, alternatively, defuzzifying durations and solving the resulting crisp problem. To do so, we simulate N possible realisations or scenarios of the problem: crisp durations for tasks are generated following a probability distribution which is coherent with the possibility distribution defined by each TFN. For each scenario $i = 1, \ldots, N$, let C^i_{max} denote the makespan obtained by executing tasks according to the ordering provided by a predictive schedule. Then, the mean ϵ-robustness of the predictive schedule, denoted $\bar{\epsilon}$, is calculated as:

$$\bar{\epsilon} = \frac{1}{N} \sum_{i=1}^{N} \frac{|C^i_{max} - C^{pred}_{max}|}{C^{pred}_{max}} \tag{11}$$

where C^{pred}_{max} is the makespan estimated by the predictive schedule. In our case, two predictive schedules are considered: the schedule obtained from solving the fuzzy problem, so C^{pred}_{max} is the expected makespan $E[C_{max}]$, and the schedule obtained from solving the defuzzified problem where TFNs are substitued by their expected value, in which case C^{pred}_{max} is a crisp makespan value.

For this robustness analysis, we concentrate on the largest problem instances, those of size 8×8 and consider, for each problem instance, $N = 1000$ deterministic instances corresponding to possible realisations. Figure 8 depicts, for each problem, the mean ϵ-robustness value of each predictive schedule, the fuzzy one (denoted $\bar{\epsilon}_F$) and the defuzzified one (denoted $\bar{\epsilon}_C$), across the N simulated scenarios. Clearly, the predictive schedule obtained from the fuzzy problem is much more robust (with smaller prediction error ϵ) than the schedule obtained from the defuzzified problem. In fact, the robustness error of the defuzzified solution is always significantly higher than that of the fuzzy solution, with error increases ranging from 28.03% to 170.51% and an average error increase of 85.04%. We may conclude that it is more robust to take into account all the available information about task durations and solve the fuzzy problem than solve the defuzzified problem.

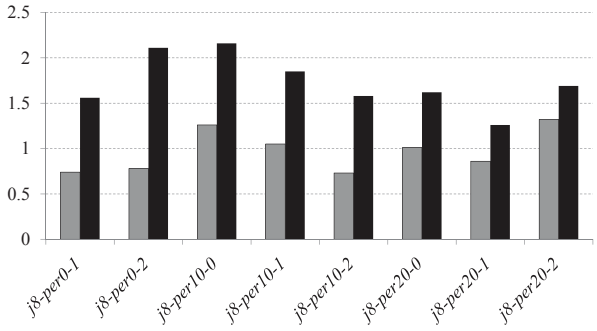

Fig. 8 Mean ϵ-robustness value of the predictive schedules obtained from the fuzzy problem ($\bar{\epsilon}_F$, in grey) and the defuzzified one ($\bar{\epsilon}_C$, in black)

.

5 Conclusions and Future Work

We have considered an open shop problem with uncertain durations modelled as triangular fuzzy numbers where the objective is to minimise the expected makespan, a problem denoted $O|fuzzp_{ij}|E[C_{max}]$. We have proposed a particle swarm optimization (PSO) method to solve this problem. An extensive experimental analysis has shown that the PSO obtains good results both in terms of relative makespan error and also in comparison to a memetic algorithm from the literature. Additionally, we have argued that it is more robust to find solutions to the fuzzy problem, taking into account the uncertainty in the durations along the scheduling process, instead of the straightforward approach of defuzzifying the durations and scheduling the resulting deterministic problem.

These promising results suggest directions for future work. First, the PSO should be tested on more difficult problems, fuzzy versions of other benchmark problems from the literature. Also, the PSO provides a solid basis for the development of more powerful hybrid methods, in combination with local search techniques, an already successful approach in fuzzy job shop problems [29]. It would also be interesting to adapt this successful PSO method to the fuzzy job shop problem.

Acknowledgments

This work Has been funded by the Spanish Ministry of Science and Education under research grants MEC-FEDER TIN2010-20976-C02-02 and MTM2010-16051 and by the Principality of Asturias (Spain) under grant Severo Ochoa BP13106.

References

1. Ahmadizar, F., Farahani, M.H.: A novel hybrid genetic algorithm for the open shop scheduling problem. International Journal of Advanced Manufacturing Technology 62, 775–787 (2012)
2. Bidot, J., Vidal, T., Laboire, P.: A theoretic and practical framework for scheduling in stochastic environment. Journal of Scheduling 12, 315–344 (2009)
3. Blum, C.: Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers & Operations Research 32(6), 1565–1591 (2005)
4. Brucker, P., Hunrink, J., Jurisch, B., Wöstmann, B.: A branch & bound algorithm for the open-shop problem. Discrete Applied Mathematics 76, 43–59 (1997)
5. Dubois, D., Fargier, H., Fortemps, P.: Scheduling under flexible constraints and uncertain data: the fuzzy approach. In: Production Scheduling, chap. 11, pp. 301–332. Wiley (2008)
6. Dubois, D., Fargier, H., Fortemps, P.: Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. European Journal of Operational Research 147, 231–252 (2003)
7. Dubois, D., Prade, H.: Possibility Theory: An Approach to Computerized Processing of Uncertainty. Plenum Press, New York (USA) (1986)
8. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions of Fuzzy Systems 7, 557–569 (1997)
9. Giffler, B., Thompson, G.L.: Algorithms for solving production scheduling problems. Operations Research 8, 487–503 (1960)
10. Gonçalves, J., Mendes, J., de M, R.M.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95 (2005)
11. González-Rodríguez, I., Palacios, J.J., Vela, C.R., Puente, J.: Heuristic local search for fuzzy open shop scheduling. In: Proceedings IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2010. pp. 1858–1865. IEEE (2010)
12. González-Rodríguez, I., Puente, J., Vela, C.R., Varela, R.: Semantics of schedules for the fuzzy job shop problem. IEEE Transactions on Systems, Man and Cybernetics, Part A 38(3), 655–666 (2008)
13. González-Rodríguez, I., Vela, C.R., Puente, J.: A memetic approach to fuzzy job shop based on expectation model. In: Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007. pp. 692–697. IEEE, London (2007)
14. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 4, 287–326 (1979)
15. Guéret, C., Prins, C.: Classical and new heuristics for the open-shop problem: A computational evaluation. European Journal of Operational Research 107, 306–314 (1998)
16. Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165, 289–306 (2005)
17. Hu, Y., Yin, M., Li, X.: A novel objective function for job-shop scheduling problem with fuzzy processing time and fuzzy due date using differential evolution algorithm. International Journal of Advanced Manufactoring Technology 56, 1125–1138 (2011)
18. Huang, Y.M., Lin, J.C.: A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems. Expert Systems with Applications 38(5), 5438–5447 (2011)
19. Kalaï, R., Lamboray, C., Vanderpooten, D.: Lexicographic α-robustness: An alternative to min-max criteria. European Journal of Operational Research 220, 722–728 (2012)
20. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Networks. pp. 1942–1948. IEEE Press, New Jersey (1995)
21. Konno, T., Ishii, H.: An open shop scheduling problem with fuzzy allowable time and fuzzy resource constraint. Fuzzy Sets and Systems 109, 141–147 (2000)
22. Lei, D.: Solving fuzzy job shop scheduling problems using random key genetic algorithm. International Journal of Advanced Manufacturing Technologies 49, 253–262 (2010)
23. Liaw, C.F.: A tabu search algorithm for the open shop scheduling problem. Computers and Operations Research 26, 109–126 (1999)

24. Liu, B., Liu, Y.K.: Expected value of fuzzy variable and fuzzy expected value models. IEEE Transactions on Fuzzy Systems 10, 445–450 (2002)
25. Noori-Darvish, S., Mahdavi, I., Mahdavi-Amiri, N.: A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due-dates. Applied Soft Computing 12, 1399–1416 (2012)
26. Palacios, J.J., Puente, J., Vela, C.R., González-Rodríguez, I.: A genetic algorithm for the open shop problem with uncertain durations. In: Proceedings of IWINAC 2009, Part I. Lecture Notes in Computer Science, vol. 5601, pp. 255–264. Springer (2009)
27. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems. Springer, third edn. (2008)
28. Puente, J., Vela, C.R., González-Rodríguez, I.: Fast local search for fuzzy job shop scheduling. In: Proceedings of 19th European Conference on Artificial Intelligence, ECAI 2010. pp. 739–744. IOS Press (2010)
29. Sha, D.Y., Cheng-Yu, H.: A modified parameterized active schedule generation algorithm for the job shop scheduling problem. In: Proceedings of the 36th International Conference on Computers and Industrial Engineering, ICCIE2006. pp. 702–712 (2006)
30. Sha, D.Y., Cheng-Yu, H.: A new particle swarm optimization for the open shop scheduling problem. Computers & Operations Research 35, 3243–3261 (2008)
31. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
32. Tavakkoli-Moghaddam, R., Safei, N., Kah, M.: Accessing feasible space in a generalized job shop scheduling problem with the fuzzy processing times: a fuzzy-neural approach. Journal of the Operational Research Society 59, 431–442 (2008)
33. Zheng, Y., Li, Y., Lei, D.: Swarm-based neighbourhood search for fuzzy job shop scheduling. International Journal of Innovative Computing and Applications 3(3), 144–151 (2011)